­National College of Ireland

Higher Diploma in Science in Cloud Computing

2014/2015

Leszek Dubicki

X14125439

leszek.dubicki@student.ncirl.ie

Cad Cloud

Technical Report



**Table of Contents**

[Executive Summary 3](#_Toc437355148)

[1 Introduction 4](#_Toc437355149)

[1.1 Background 4](#_Toc437355150)

[1.2 Aims 4](#_Toc437355151)

[1.3 Technologies 5](#_Toc437355152)

[1.4 Structure 5](#_Toc437355153)

[2 System 6](#_Toc437355154)

[2.1 Requirements 6](#_Toc437355155)

[2.1.1 Functional requirements 6](#_Toc437355156)

[2.1.2 Environmental requirements 7](#_Toc437355157)

[2.1.3 Usability requirements 7](#_Toc437355158)

[2.2 Design and Architecture 8](#_Toc437355159)

[2.3 Implementation 10](#_Toc437355160)

[2.3.1 Server components 10](#_Toc437355161)

[2.3.2 Main python client component 12](#_Toc437355162)

[2.3.3 Python COM server acting as CC client 13](#_Toc437355163)

[2.3.4 VBA component 14](#_Toc437355164)

[2.4 Testing 16](#_Toc437355165)

[2.5 Graphical User Interface (GUI) Layout 17](#_Toc437355166)

[2.5.1 Server web page 17](#_Toc437355167)

[2.5.2 SolidWorks Feature User Interface 20](#_Toc437355168)

[3 Conclusions 27](#_Toc437355169)

[4 Further development or research 28](#_Toc437355170)

[5 References 29](#_Toc437355171)

[6 Appendix 30](#_Toc437355172)

[6.1 Project Proposal 30](#_Toc437355173)

[6.2 Requirements Specification 30](#_Toc437355174)

[6.3 Monthly Journals 30](#_Toc437355175)

[6.4 CD with files containing source code 30](#_Toc437355176)

# Executive Summary

The CADCloud project is a client server system created to automate collaboration and sharing of project information between designers and other employees working on the same project. It is aimed to be a distributed application with the server component residing in the network and client components using server resources over the network via HTTP protocol. All source code of components were developed using either Python or VBA programming language, but other client applications using the CADCloud API can be written in any programming language as long as they communicate with the server over a network or with the client through COM interface in Microsoft Windows environment. The server data can be managed directly using html user interface or from client application. The main client application in this project was written as a feature that can be embedded into any SolidWorks document and connect this document to the CADCloud service available on the network. The application can be also used as a base for more specialised application sharing custom engineering data.

# Introduction

## Background

Currently working with modern CAD systems means working with files of their native file format stored in version control environment usually supplied by the same software vendor. An examples would be Solidworks and EPDM or Creo Parametric and Windchill. Data sharing across multiple files is usually achieved by creating external references to other native files or by linking to other popular format files, usually excel.

CAD systems like Solidworks or CATIA or Creo Parametric are very powerfull modelling tools but also very expensive systems. There are also free alternatives, usually with less capabilities than commercial systems. Having cloud based platform to share cad data creates opportunity to adopt both commercial and open software and have engineering teams working and cooperating in one environment without a need to purchase large quantities of very expensive licenses.

Other possible use of such platform would be connecting CAD applications to ERP systems. Such systems exist at the moment but require purchasing additional licenses (for example Solidworks ToolWorks ERP-Link or SOLIDWORKS® Enterprise PDM integration for ERP).

## Aims

The project aimes to create platform and initial components for sharing engineering information between applications typically used in engineering enterprise. It includes RESTfull API that allows connecting new components and CAD feature/plugin embedded into popular Solidworks.

Data that will be shared include text and numerical values (to be stored as global variables or custom properties in CAD files). In the future it will also allow to share basic 2D geometry (points, lines, arcs and circles) stored as sketches or curves in CAD files. This data will be either synchronized with connected cloud service (ie will refresh on every rebuild of cad file) or will be refreshed only on user request.

## Technologies

The CADCloud is a client-server system. Server is a REST-full API developed using Python programming language and Flask framework. Client component was developed for Solidworks CAD system in VBA programming language embedded into Solidworks, using MacroFeature interface.

Below is the list of technologies used in the project:

* Python programming language
* Visual Basic for Applications (VBA)
* Flask framework
* Python SQLALCHEMY
* flask-SQLALCHEMY
* BOOTSTRAP
* Python WTForms
* Flask-Migrate
* SQLite
* MySQL
* json
* Component Object Model (COM)
* Python for Windows Extensions (pyWin32)

# System

The system must release CAD designer from importing static data from external sources and provide tool to share data online between different systems.

## Requirements

The basic requirement for the system is to be easy to use and transparent for the designer. It means that custom CC feature would be inserted into the tree view like any other objects. This CC feature would be easily configurable (server instance and login details must be provided, than available resources of the server must be presented for the user)

### Functional requirements

The most recent Use-Case Diagram is presented on Figure 1. Since in typical engineering enterprise environment users can’t (and shouldn’t) register themselves users access rights are managed by the system administrator.

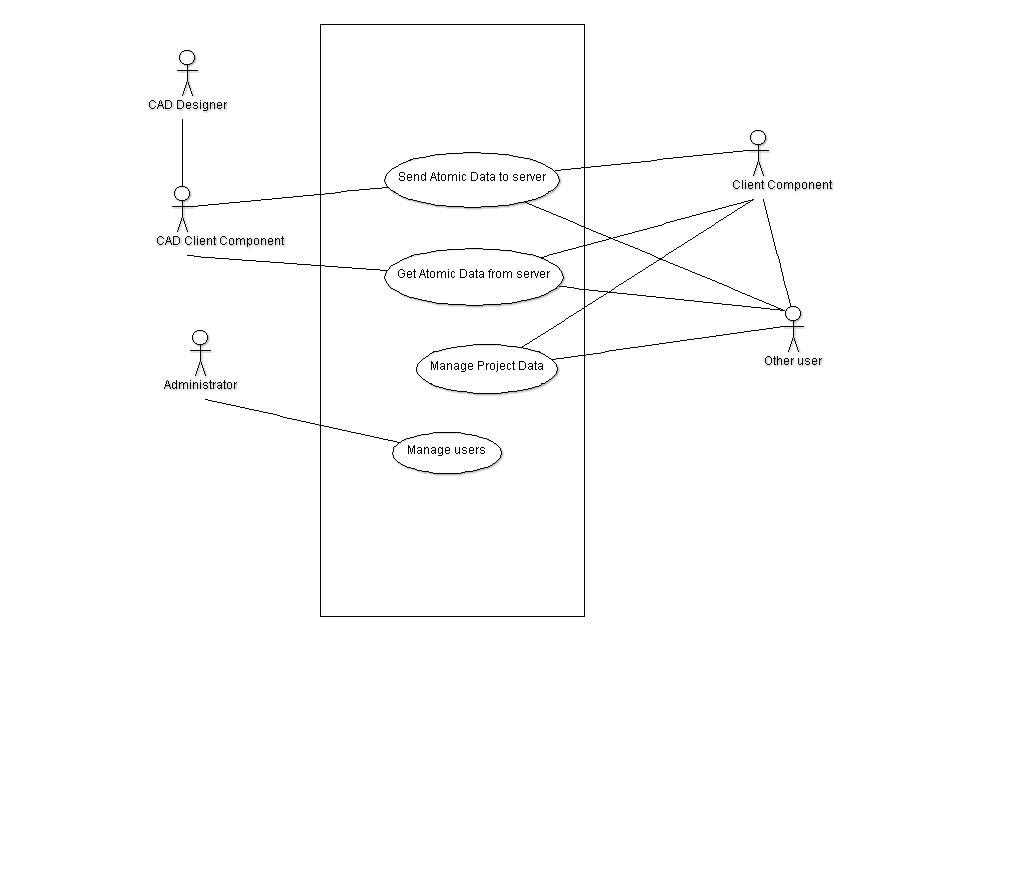


Figure CADCloud System Use-Case Diagram

### Environmental requirements

The system doesn’t require any special environment settings. Server component can run on any Unix or Windows server with Python interpreter, Client systems should have Python interpreter installed (for SolidWorks client of course SolidWorks must be installed as well)

### Usability requirements

Because the System is designed to work with very fast and efficient CAD systems and due to the fact that parts/assemblies are rebuilt / recalculated many times during the work day (practically every minute) the response time for the System must be very short. The system was tested in standard home network environment (40Mbps download / 25Mbps upload speed) and working with cad documents was efficient enough (no difference between documents with and without CC features). However – for very large assemblies including many parts having cc features each the server response time might be too slow for efficient design work. In case the rebuild time becomes too low embedded CC features can be set to synchronize manually, in this case the System response is not that critical. Also in future versions of the system optimization techniques will be implemented in order to shorten the response time and decrease network load.

## Design and Architecture

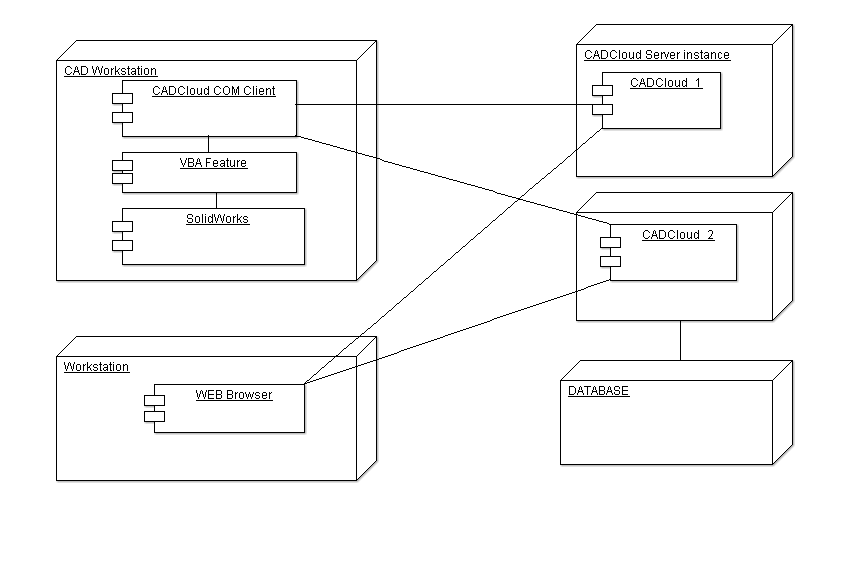


Figure CADCloud system deplyment diagram

CADCloud application is designed to have a client-server structure, usually with one server instance (can be more than that) and several clients connected to it. The server provides a platform for data exchange between CAD applications (CAD to CAD and CAD to other apps).

The server component is developed as a REST-full API using Python with flask framework and SQLAlchemy (flask-SQLAlchemy) extension to manage database instances. The API provides entry points (URI-s) listed in Table 1. The server data is designed to be easily managed using web browser – for instance by Project Manager or Quality specialist responsible for some aspects of the project. The client application (currently only SolidWorks add-in component is developed) can write or read any data from the server application using api entries and json data interchange format. Client component is implemented in Python using requests module and python for windows pywin32 extensions to make CADCloud resources available for other applications through COM interface. Solidworks component is written in VBA programming language and is communicating with the CADCloud client component instance running on the same system through standard Microsoft Windows COM interface.

Table List of Uri's for CadCloud server API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HTTP Method | URI | Action | Parameters | Pre/Post conditions |
| GET | / | Retrieve the main page of the application (for json –retrieve only information that the service is available) | None | None / None |
| GET | /cad/api/v0.1/projects | Retrieve the list of all projects with all project data | None | None / None |
| GET | /cad/api/v0.1/projects\_list | Retrieve only the list of only project numbers in json, redirect to “/cad/api/v0.1/projects” for html | None | None / None |
| GET | /cad/api/v0.1/projects/<int:project\_id> | get project with given id | None | project exists / None |
| GET | /cad/api/v0.1/projects\_by\_num/<string:project\_number> | get project with given project number | None | project exists / None |
| GET, POST | /cad/api/v0.1/projects/add | add new project | None | None / new project created |
| GET, POST, PUT | /cad/api/v0.1/projects/edit/<int:project\_id> | edit project with given id | None | project exists / project updated |
| GET, POST | /cad/api/v0.1/projects/delete/<int:project\_id> | delete project with given id | None | project exists / project deleted |
| GET, POST | /cad/api/v0.1/variables/add/<int:project\_id> | add variable to project with given id | None | project exists / new variable created |
| GET, POST, PUT | /cad/api/v0.1/variables/edit/<int:project\_id>/<string:variable\_type>/<int:variable\_id> | edit variable with given id and type within given project | None | variable exists / variable updated |
| GET, POST, PUT | /cad/api/v0.1/variables/edit/<int:project\_id>/<string:variable\_name> | edit variable of given name within given project | None | variable exists / variable updated |
| GET | /cad/api/v0.1/variables/<int:project\_id> | get list of variables within project | None | project exists / None |
| GET | /cad/api/v0.1/get\_variable/<int:project\_id>/<string:var\_name> | get variable within a project by name | None | project and variable exist / None |
| GET, POST | /cad/api/v0.1/variables/delete/<int:project\_id>/<string:variable\_type>/<int:variable\_id> | delete variable with given id of given type within project with given project id | None | variable exists / variable deleted |

## Implementation

The server component is designed using standard Model View Controller architecture using Python Flask with flask-SQLALCHEMY extension for managing data Models. The Entity Relationship Diagram for models is presented on Figure 3 below.

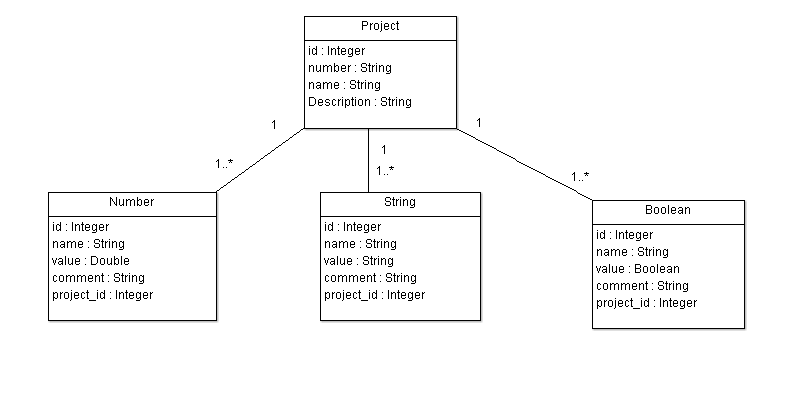


Figure Entity Relationship Diagram of the project data models

### Server components

Main models and forms definitions are placed in file ccapp.py. It can be than imported in any other python file containing web application entry points. Example model definition in ccapp.py (using flask and flask-SQLALCHEMY modules) looks as follows:

**from** flask\_sqlalchemy **import** SQLAlchemy

**from** flask **import** Flask

app **=** Flask**(**\_\_name\_\_**)**

db **=** SQLAlchemy**(**app**)**

#Model Definition:

**class** **Project(**db**.**Model**):**

#class for projects record

#\_\_tablename\_\_ = "projects"

id **=** db**.**Column**(**db**.**Integer**,** primary\_key**=True)**

name **=** db**.**Column**(**db**.**String**(**128**))**

project\_number **=** db**.**Column**(**db**.**String**(**32**),** index **=** **True)**

numbers **=** db**.**relationship**(**'Number'**,** backref**=**'project'**,** lazy**=**'dynamic'**)**

strings **=** db**.**relationship**(**'String'**,** backref**=**'project'**,** lazy**=**'dynamic'**)**

booleans **=** db**.**relationship**(**'Boolean'**,** backref**=**'project'**,** lazy**=**'dynamic'**)**

description **=** db**.**Column**(**db**.**String**(**512**))**

**def** \_\_init\_\_**(**self**,** name**,** number**,** description **=** ""**):**

self**.**name **=** name

self**.**project\_number **=** number

self**.**description **=** description

**def** \_\_repr\_\_**(**self**):**

**return** "<Project %r>" **%** self**.**project\_number

For creating and editing models data in html forms WTForms module is used. An example forms definition is presented below (notice that form class definition of class ProjectAddForm is inherited from ProjectEditForm and that additionally to standard WTForms validators custom validation function was used to ensure that project number is unique):

##forms definitions:

**class** **ProjectEditForm(**Form**):**

name **=** TextField**(**'Project Name'**,** description**=**'Enter Name of the project here'**,**validators**=[**Required**()])**

description **=** TextAreaField**(**u'Project Description'**,** **[**validators**.**optional**(),** validators**.**length**(**max**=**200**)])**

submit\_button **=** SubmitField**(**'Update Project'**)**

**class** **ProjectAddForm(**ProjectEditForm**):**

project\_number **=** TextField**(**'Project Number'**,** description**=**'Number of the project'**,** validators**=[**Required**()])**

submit\_button **=** SubmitField**(**'Create Project'**)**

**def** validate**(**self**):**

rv **=** Form**.**validate**(**self**)**

**if** **not** rv**:**

**return** **False**

#look for a project with given number (must be unique)

pro **=** findProject**(**self**.**project\_number**.**data**)**

**if** pro **==** **None:**

**return** **True**

**else:**

self**.**project\_number**.**errors**.**append**(**'Project with given number already exists!'**)**

All application routes are defined in file ccserver.py. For example start page route is defined as follows (notice that if ‘application/json’ header is present application just returns short information that the service is available):

#Start Page:

@app.route**(**'/'**)**

**def** start\_page**():**

**if** request**.**headers**[**'Content-Type'**]** **==** 'application/json'**:**

#basically do nothing - just let know that it's up:

**return** jsonify**({**'cadcloud'**:**'up'**})**

**else:**

**return** render\_template**(**'index.html'**,** title**=**"CADCloud"**,** active **=** 'home'**,** get\_active **=** get\_active**)**

Below is also example of more complex route – for information about particular project:

@app.route**(**'/cad/api/v0.1/projects/<int:project\_id>'**)**

**def** get\_project**(**project\_id**):**

#retrieves only the project with a given id and returns it's html or json representatoin

project **=** Project**.**query**.**get**(**project\_id**)**

**if** request**.**headers**[**'Content-Type'**]** **==** 'application/json'**:**

#get json object with project data:

P2 **=** Project**.**query**.**get**(**project\_id**)**

pro **=**dict\_model**(**P2**)**

**return** jsonify**({**'project'**:**pro**})**

**else:**

**return** render\_template**(**'show\_project.html'**,**project **=** project**,** active **=** ''**,** get\_active **=** get\_active**)**

### Main python client component

Client Python component was implemented using python requests module which provides easy to use http interface. An example below shows class definition with methods retrieving information from CadCloud web server (from file ccclient.py):

#client app to connect to ccloud server app

**import** requests

**class** **CCloud:**

#class to perform operations on single CADCloud web app (given by base URL)

**def** \_\_init\_\_**(**self**):**

self**.**\_url **=** **None**

self**.**\_headers **=** **{**'content-type'**:** 'application/json'**}**

**def** set\_url**(**self**,** url**):**

self**.**\_url **=** url

**def** setUrl**(**self**,** url**):**

#copy of set\_url method

self**.**set\_url**(**url**)**

**def** getUrl**(**self**):**

**return** self**.**\_url

**def** checkUrl**(**self**):**

#checks if there is something under given url, if not then return False

**if** self**.**\_url **==** **None** **or** self**.**\_url **==** "None"**:**

**return** **False**

**else:**

#give it a chance:

**try:**

r **=** requests**.**get**(**self**.**\_url**,** headers **=** self**.**\_headers**)**

**if** **(**r**.**status\_code **==** 200**)** **and** **(**'cadcloud' **in** r**.**json**())** **and** **(**r**.**json**()[**'cadcloud'**]** **==** 'up'**):**

**return** **True**

**else:**

**return** **False**

**except:**

**return** **False**

**def** get\_projects**(**self**):**

#gets list of projects

uri **=** '/cad/api/v0.1/projects'

**if** self**.**checkUrl**()** **==** **True:**

r **=** requests**.**get**(**self**.**\_url **+** uri**,** headers **=** self**.**\_headers**)**

**else:**

#return custom status code

**return** **{**'status\_code'**:**1404**,** 'projects'**:** **[]}**

**if** r**.**status\_code **==** 200**:**

**return** r**.**json**()**

**else:**

**return** **{**'status\_code'**:**r**.**status\_code**,** 'projects'**:** **[]}**

#I'd rather not store all the projects not to consume too much memory (see below)

**def** getProjectNumbers**(**self**):**

#only a list of project numbers (without id's or anything)

projects **=** self**.**get\_projects\_list**()**

pList **=** **[]**

**for** p **in** projects**[**'projects'**]:**

pList**.**append**(**p**)**

**return** pList

### Python COM server acting as CC client

To allow other windows programs easily connect to CC server resources special component with definition of COM server was created (file cccomclient.py). This component defines special class which will be accessible through COM and must be registered in the system. Below the class definition (notice it inherits from CClient class therefore has all methods necessary to communicate with ccserver instance):

#CClient COM component

**import** pythoncom**,** os**.**path

**import** ccclient

**from** ccclient **import** CCStore

#filename with machine-specific config (for now only \_reg\_clsid\_ is written there)

fName **=** '.ccguidfile.txt'

**class** **CCComClient(**CCloud**):**

#below clsid definition was turned off since it's machine specific,

#example \_reg\_clsid\_ = "{E51C6BDA-7811-45B3-B962-AADDBDF0E2C3}"

#\_reg\_clsid\_ can be generated with pythoncom.CreateGuid()

#in this case it's either generated using above or taken from file writen in variable fName (see above)

\_reg\_clsid\_ **=** ""

\_reg\_desc\_ **=** "Python Cad Cloud Client COM Server"

\_reg\_progid\_ **=** "Python.CadCloud"

\_public\_methods\_ **=** **[**"getProjectNumbers"**,** "setUrl"**,** "getUrl"**,** "getprojectbynumber"**,** "getvariable"**,** "get\_variable"**]**

\_public\_attrs\_ **=** **[]**

\_readonly\_attrs\_ **=** **[]**

**def** \_\_init\_\_**(**self**):**

CCStore**.**\_\_init\_\_**(**self**)**

**def** getprojectbynumber**(**self**,** project\_number**):**

#retrieve project, but instead of dictionary return list (which will be accesible as variant array in VBA)

project **=** CCStore**.**get\_project\_by\_number**(**self**,** project\_number**)**

#project = {}

#test dictionary similar to the one that should be returned:

#project = {'project':{'id':9, 'project\_number':'cykkk', 'name':'pykkk', 'description':'hehehehe'}}

projectList **=** **[]**

**if** 'status\_code' **in** project **and** project**[**'status\_code'**]** **>=** 300**:**

projectList**.**append**(**'project\_not\_found'**)**

**elif** **(**'project' **in** project**)** **and** **(not** project**[**'project'**]** **==** **None):**

#building a list of project data:

p **=** project**[**'project'**]**

projectList**.**append**(**p**[**'id'**])**

projectList**.**append**(**p**[**'project\_number'**])**

projectList**.**append**(**p**[**'name'**])**

projectList**.**append**(**p**[**'description'**])**

#change all None values to empty strings since vba has problem with them:

**for** i **in** range**(**0**,** len**(**projectList**)-**1**):**

**if** projectList**[**i**]** **==** **None:**

projectList**[**i**]** **=** ""

**else:**

projectList**.**append**(**'project\_not\_found'**)**

**return** projectList

Please also notice that resources data are returned as lists, not dictionaries like in CClient class. The reason for that is that VBA doesn’t have dictionary data type and that pywin32 module converts lists into arrays of Variant data type which is directly mapped into VBA Variant array.

After definition the class must be registered in the system as follows:

**if** \_\_name\_\_**==**'\_\_main\_\_'**:**

#create COM server id:

\_reg\_clsid\_ **=** guid**()**

CCComClient**.**\_reg\_clsid\_ **=** \_reg\_clsid\_

**import** win32com**.**server**.**register

#get guid for this machine:

win32com**.**server**.**register**.**UseCommandLine**(**CCComClient**)**

This code registers class CCComClient as a COM server component available for clients within the system. The file cccomclient.py must be executed in the system in order to be available to other processes as a COM server.

VBA component

### VBA component

To call an instance of CCComClient class in VBA inside SolidWorks the following code must be executed:

'declaration of the CC object variable:

Public CC As Object

'creation of object instance:

Set CC **=** CreateObject**(**"Python.CadCloud"**)**

All VBA code is placed in file macro\_feature-ccloud.swp. The project window in Visual Basic editor embedded in SolidWorks is shown on Figure 4.

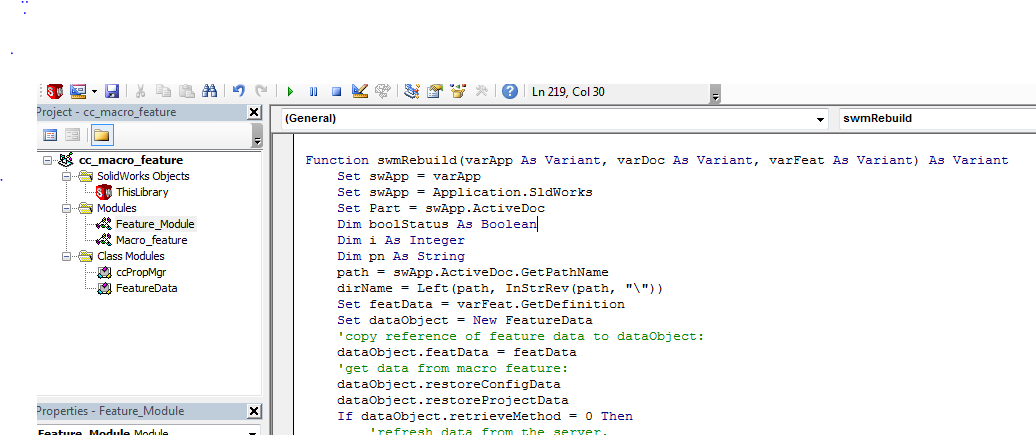


Figure VBA project stored in file macro\_feature-ccloud.swp

The cc\_macro\_feature project includes two standard modules:

* Macro\_feature with main() method which is executed as the first method and which creates macro feature in SolidWorks features tree.

Below there is an example of the method that has to be executed to create macro feature in SW document:

'definition of objects:

Dim swApp As SldWorks.SldWorks

Dim swModel As SldWorks.ModelDoc2

Dim swFeatMgr As SldWorks.FeatureManager

Dim swFeat As Feature

'initialization of objects:

Set swApp **=** Application.SldWorks

Set swModel **=** swApp.ActiveDoc

Set swFeatMgr **=** swModel.FeatureManager

Set swFeat **=** swFeatMgr.InsertMacroFeature3**(**BaseName**,** ProgId**,** vMacroMethods**,** \_

paramNames**,** paramTypes**,** paramValues**,** DimTypes**,** DimValues**,** EditBodies**,** \_

IconFiles**,** options**)**

where:

* BaseName is String with feature name suffix
* vMacroMethods – array of macro methods to be used in the feature
* paramNames, paramTypes, paramValues – arrays of parameters values, names and types
* The ProgId, , DimTypes, DimValues, EditBodies, IconFiles and options arguments are either not required for VBA code or are empty values in this case
* Feature\_Module with functions that are executed when the existing macro feature is either being rebuilt (method swmRebuild) or when definition of the feature is edited (method swmEditDefinition), both defined when creating the macro feature (see above)
* ccPropMgr class module where all User Interface controls are defined
* FeatureData class module where all data exchange between CCComclient class, Solidworks document and feature data stored in feature object is defined. An example code for restoring data from the macro feature is presented below:

'macro feature data object definition:

Private pFeatData As SldWorks.macroFeatureData

'pFeatData seter method:

Public Property Let featData**(**c As Object**)**

Set pFeatData **=** c

End Property

'method to restore all data from macro feature:

Public Sub restoreConfigData**()**

'get data saved in macro feature and writh them into global variables:

pFeatData.GetStringByName "url"**,** pUrl

pFeatData.GetStringByName "project\_number\_custom\_info"**,** pProSelCustomInfo

pFeatData.GetIntegerByName "retrieve\_method"**,** pRetrieveMethod

pFeatData.GetIntegerByName "project\_number\_selection\_method"**,** pProSelMethod

pFeatData.GetIntegerByName "project\_number\_folder\_direction"**,** pProSelPathDirection

pFeatData.GetIntegerByName "project\_number\_folder\_level"**,** pProSelPathLevel

pFeatData.GetStringByName "project\_number\_custom\_info"**,** pProSelCustomInfo

End Sub

## Testing

For testing purpose Solidworks version 2014 was used. There were several Part models created with CADCloud feature in their design tree and tested with all possible settings. Also an instance of CC Server was created on Okeanos Virtual Machine with Linux Ubuntu system. All developed functionality worked well after implemented corrections and improvements.

## Graphical User Interface (GUI) Layout

There are two aspect of User Interface in the system – server GUI which is implemented in html using Flask Python framework and Bootstrap framework, and client User Interface implemented in specific CAD system or other application that runs locally. Server GUI is common for all users, unlike client GUI which is specific to client CAD application in which it’s implemented. Currently only SolidWorks platform is supported by Cad Cloud system.

### Server web page

Server interface consists of:

a) the main page

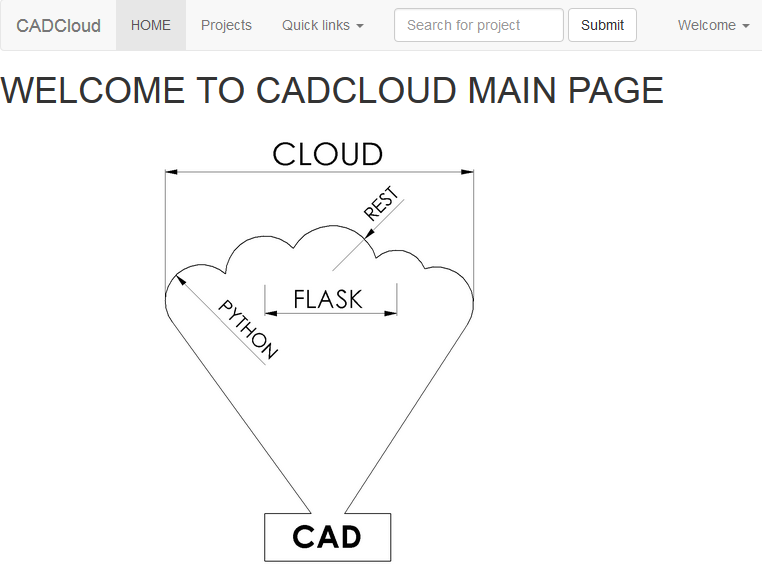


Figure CadCloud Server UI start page

b) Projects listing page:

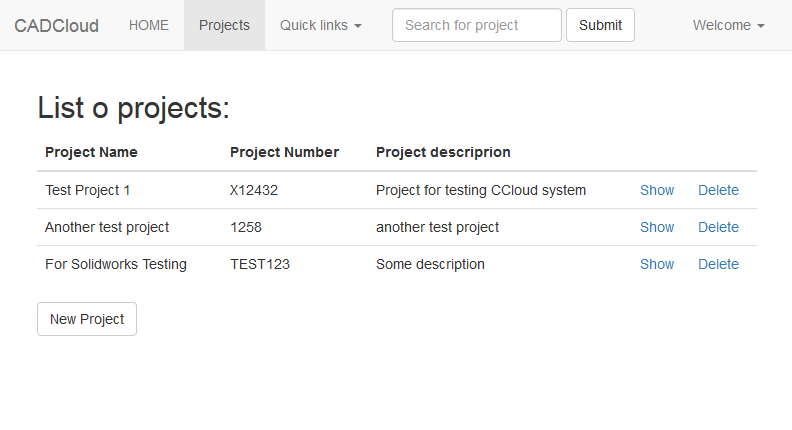


Figure Projects listing page

c) Project creation form or project edit form:

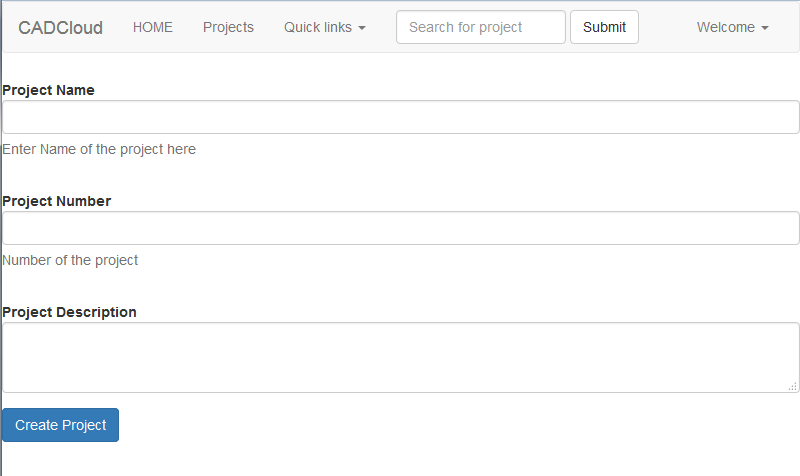


Figure New project creation form

d) Projects pages with variables list for each project:

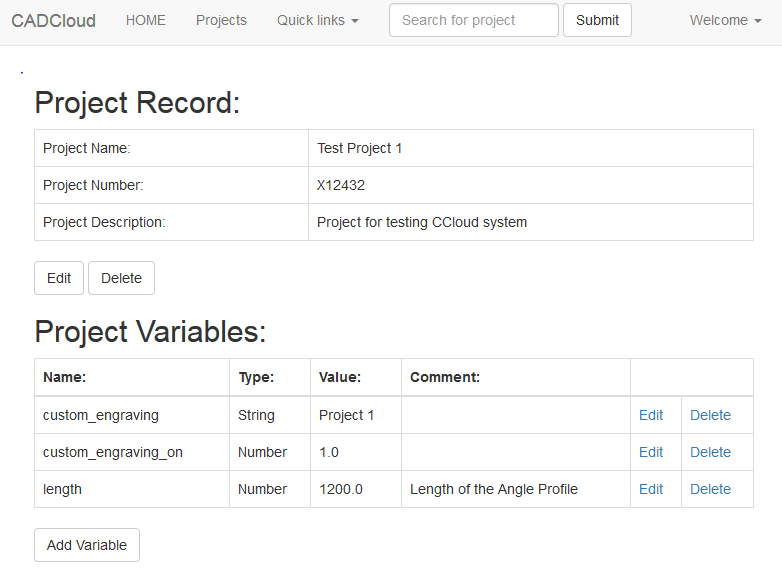


Figure Project record together with variables list

e) Variable creation / edit form:

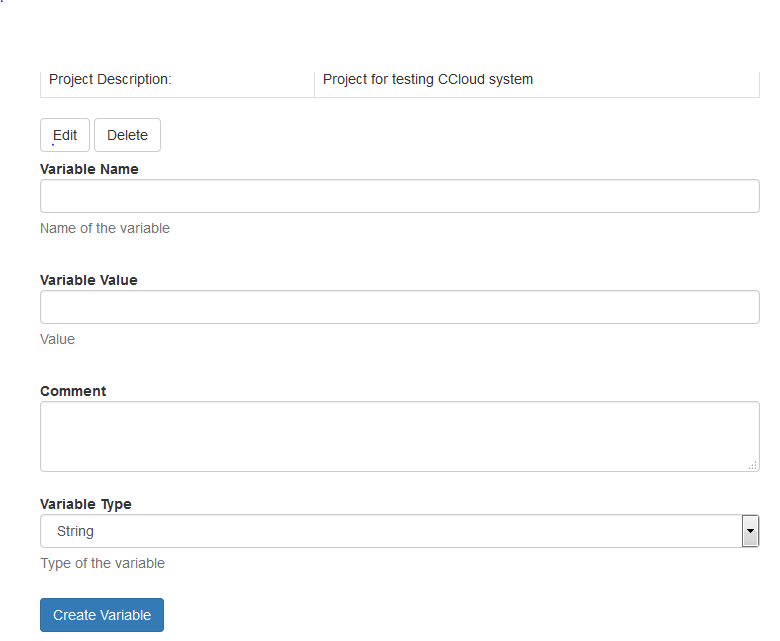


Figure New variable creation form

### SolidWorks Feature User Interface

Very important part of the whole project is user interface of features embedded into client cad applications because this is the part that will be used the most often by the end users. There are two main options to create user interface for SolidWorks macro features – standard vba forms and SolidWorks-specific user interface called IPropertyManagerPageControl Interface. For purpose of this project the second option was chosen due to the fact that it’s creating user interface identical as interface of standard SolidWorks features like Extrudes or Patterns and therefore will feel more natural for the end users.

To insert macro feature into the solidworks document file “macro feature-ccloud.swp” must be executed in SolidWorks (\*.swp and \*.swb files are standard files containing VBA code used in SolidWorks. They are binary files and can’t be edited using standard text editors, only through VBA editor available in each SolidWorks installation). After inserting the feature will appear in Feature Tree:

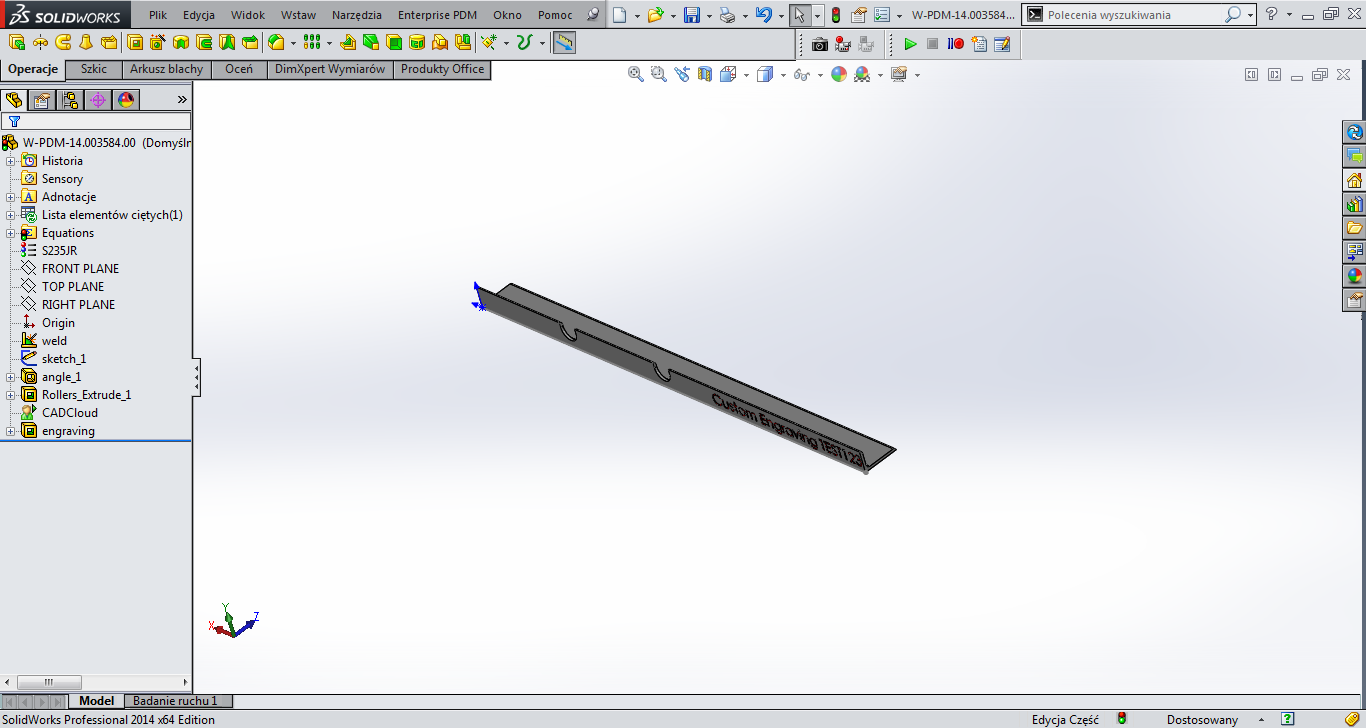


Figure SolidWorks graphical window with features tree on the left

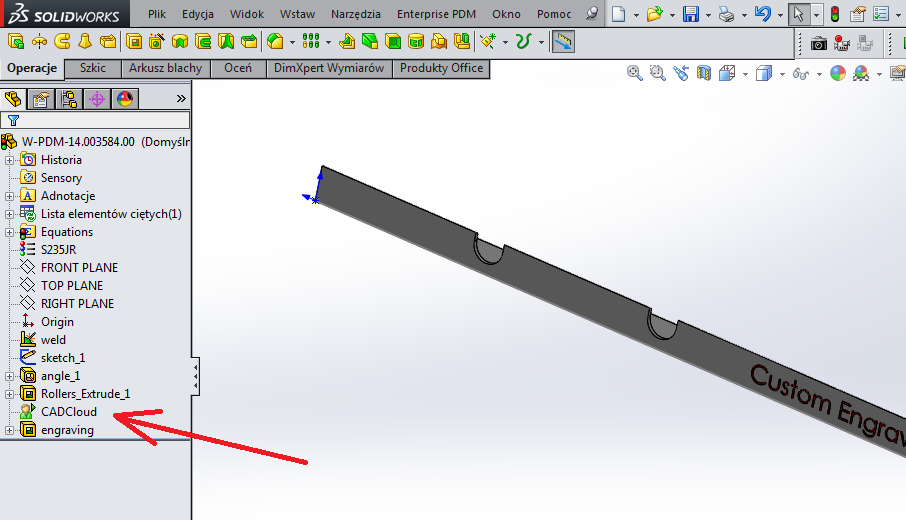


Figure CadCloud feature in SolidWorks features tree

When CC feature is in features tree it’s possible to edit it’s definition the standard SolidWorks way – by clicking it and then clicking edit-definition icon:

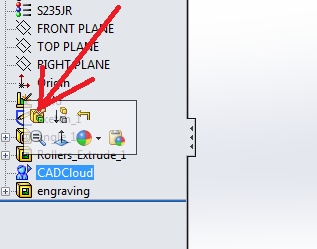


Figure Context menu with edit definition button

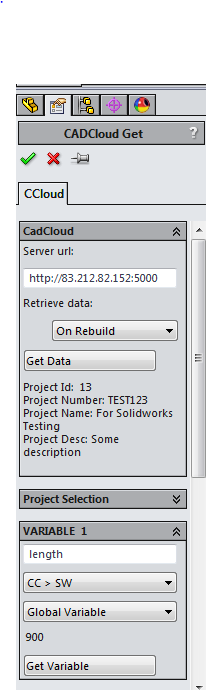


Figure CadCloud edit definition user interface page with server settings on the top

After clicking edit definition icon the Cad Cloud user interface will appear. The interface is divided into several groups:

* server settings
* project selection settings
* five variable settings groups

Each group can be expanded or collapsed by clicking on arrow in top right corner

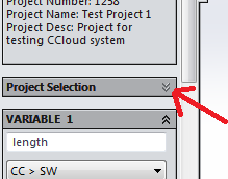


Figure Collapsed Project Selection controls group

In server settings group there are two important settings to edit – Cad Cloud server url and information retrieval setting – it can be set either to “On rebuild” or “On edit definition”. Below is button to manually retrieve all data at any time during editing feature definition, and project information got from the server.

The next group is project selection group. Project is retrieved from the server using project number. There are three ways to select project number:

a) By retrieving list of projects from the server and manually selecting one of them. This is the easiest method to use one time but it’s probably the least likely to being used in practice applications because it requires selecting project for each individual document weather it was created from scratch or copied from other reference project.

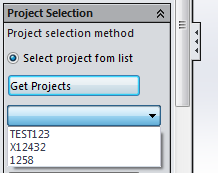


Figure Project Selection from list

b) Using selected custom property as a source for the project number. Custom Properties SolidWorks file properties that have <key>, <type>, <value> structure and are used to write custom information in the file (usually things like engraving details, surface finish, customer name, or - project number). This method can be automated for example by using certain Product Data Management systems (like SolidWorks Enterprise PDM based on Conisio PLM) by setting this custom property automatically for files created in the repository based on any conditions, for example path of the folder where the file was created.

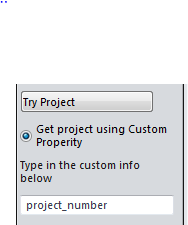


Figure Project selection using custom property

c) Using folder name where the file is placed. This is the most common way of organizing the structure of any documents within most enterprises and for companies using CAD software the folder structure is usually very much standardised (it’s necessary when dealing with hundreds thousands of CAD files). In CadCloud Solidworks feature folder containing project number can be selected by setting distance from the beginning or from the end of the path (for example on Figure 1 the file path is <D:\projects\nci\CADCloud\test\X12432\Design\Solidworks> 5-th folder from the beginning of the path is selected as the project number folder). If this file was copied to another folder within the same structure (for example <D:\projects\nci\CADCloud\test\TEST123\Design\Solidworks>) – then the project number will automatically change to TEST123.

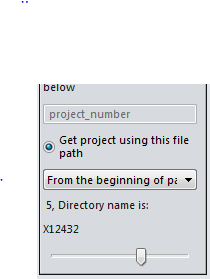


Figure Project selection using folder

Finally at the bottom part of CadCloud there are five groups for defining project variables to exchange with the server (Figure 14). Below textbox with variable name there is ComboBox where direction of communication is set, and beneath it there is a place where to put or look for the variable value (it’s either global variable placed in SolidWorks Equations or file Custom Property)

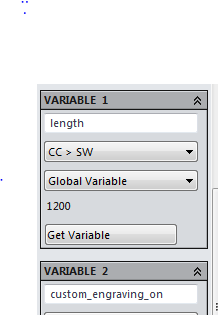


Figure Variables definition groups

If there are more than 5 variables to take from the server more CadCloud features can be put in the document. Every new feature might also use different server instance in case resources from more than one services are required.

# Conclusions

Although not all points from Requirement Specification were fulfilled the application works very well and can be a very good base for more complex and more complete system that might be very useful in many engineering enterprises.

Typical use of a system would be building custom web API that would provide services used directly inside CAD application, which would be the whole new technology in mechanical engineering software.

# Further development or research

With more resources, the presented project could have big potential to become more complex CAD data sharing system. An example is sharing 3D geometry between applications, or even a logic (transfer code that would be executed in CAD environment). Although lot of effort is required to transform presented project into mature robust technology useful in the industries where big money are involved.

The system could be very useful in companies which develop big amount of projects, each of them in isolation from each other – it would improve data coordination between Sales, Design, Manufacturing Quality departments.

The system might also be very useful in big-project environment with very big amount of data that needs to be shared among different design teams (e.g. R&D team working on product and very often releasing new revision and production equipment team that needs to analyze changes and adapt their models/drawings for every new revision of product).

# References

*CADSharp - Create Custom Features with SolidWorks API*. (n.d.). Retrieved from http://www.cadsharp.com/blog/the-solidworks-api-crown-jewel-macro-features/

*Flask-Migrate*. (n.d.). Retrieved from https://flask-migrate.readthedocs.org/en/latest/

*Flask-SQLAlchemy*. (n.d.). Retrieved from http://flask-sqlalchemy.pocoo.org/2.1/

*Flask-WTF*. (n.d.). Retrieved from https://flask-wtf.readthedocs.org/en/latest/

*IPropertyManagerPageControl Interface*. (n.d.). Retrieved from http://help.solidworks.com/2015/English/api/sldworksapi/SolidWorks.Interop.sldworks~SolidWorks.Interop.sldworks.IPropertyManagerPageControl\_members.html

*Overview of Macro Features*. (n.d.). Retrieved from http://help.solidworks.com/2013/English/api/sldworksapiprogguide/Macro\_Features/Overview\_of\_Macro\_Features.htm

*Python Flask*. (n.d.). Retrieved from http://flask.pocoo.org/

*Python for Windows extensions*. (n.d.). Retrieved from http://docs.activestate.com/activepython/2.6/pywin32/PyWin32.HTML

*Requests: HTTP for Humans*. (n.d.). Retrieved from http://docs.python-requests.org/en/latest/index.html

*Solidworks API Help*. (n.d.). Retrieved from http://help.solidworks.com/2016/English/api/sldworksapiprogguide/Welcome.htm

*SQLAlchemy*. (n.d.). Retrieved from http://www.sqlalchemy.org/

# Appendix

## Project Proposal

## Requirements Specification

## Monthly Journals

Monthly Journal #1

## CD with files containing source code